Software Engineering

Ajanta
Page No.
Date

Software — Program

↓ ↓

It is a collection of program (A group of progs) large Docs

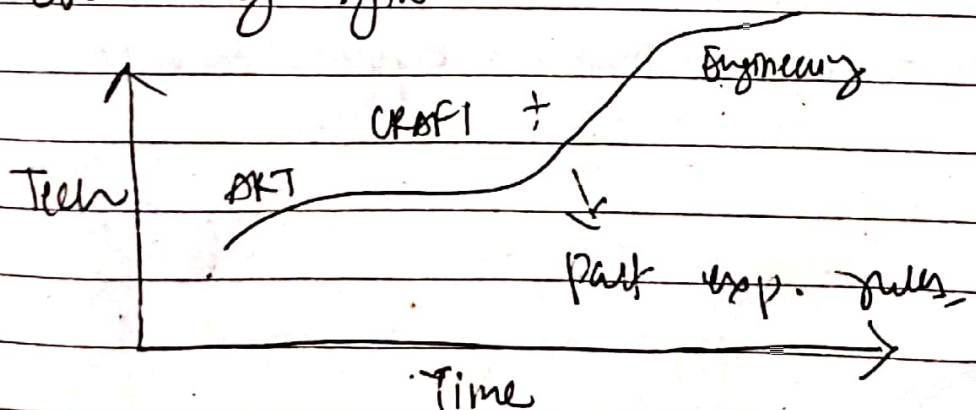
Set of instructions (certain tasks for program) no ~~does~~ docs

Engineering - developing systematically
→ cost effective
less resources & less time } Successfull Software

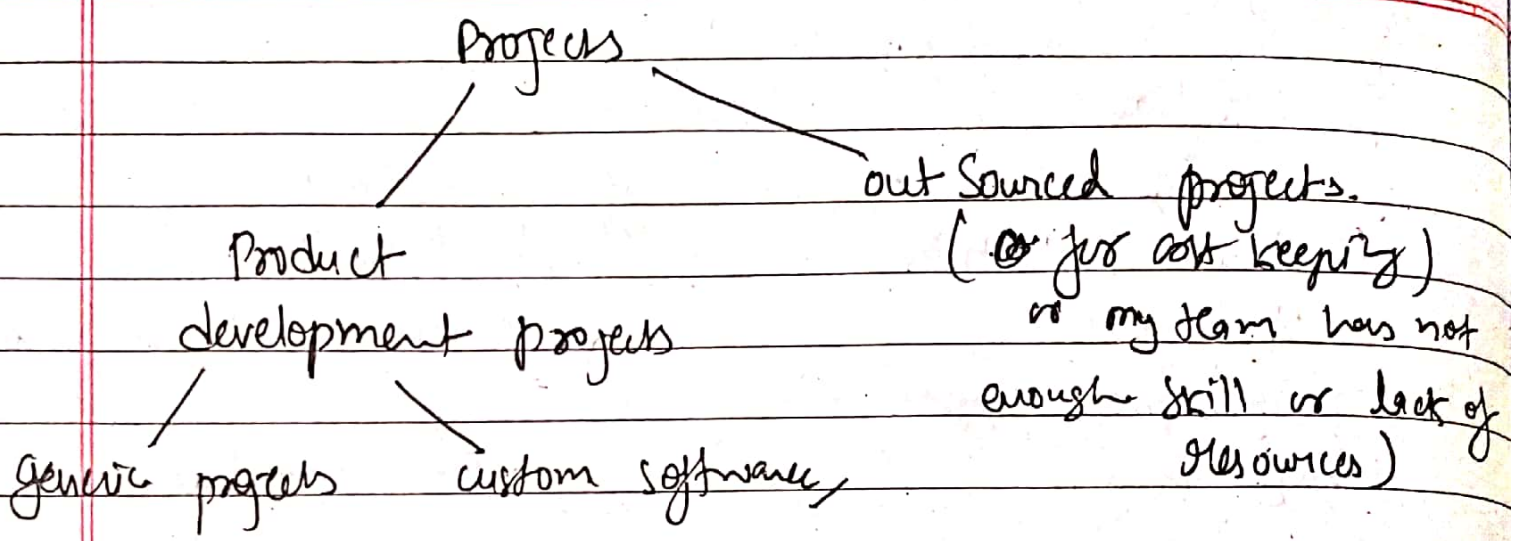
Software failure- #
If a software ~~m~~ doesn't meet unexpected situations then we say it as software failure.
& it ~~also~~ should also support technical upgradations.

Software ware engineering is a systematic & ~~cost~~ cost effective way of developing a software.

Evolution of Software



Types of S/w Development



Why to study software Engineering

→ Skill to develop large software products
→ team work
→ complex software
→ effectively
→ abstractions..

Software life cycle models

Software life cycle:-

Inception - Design - Devlop - deployed - maintainence - disposed

SDLC

Software Development life cycle

Requirements → Design → Development/coding → Testing → maintainence

(source code, test reports)

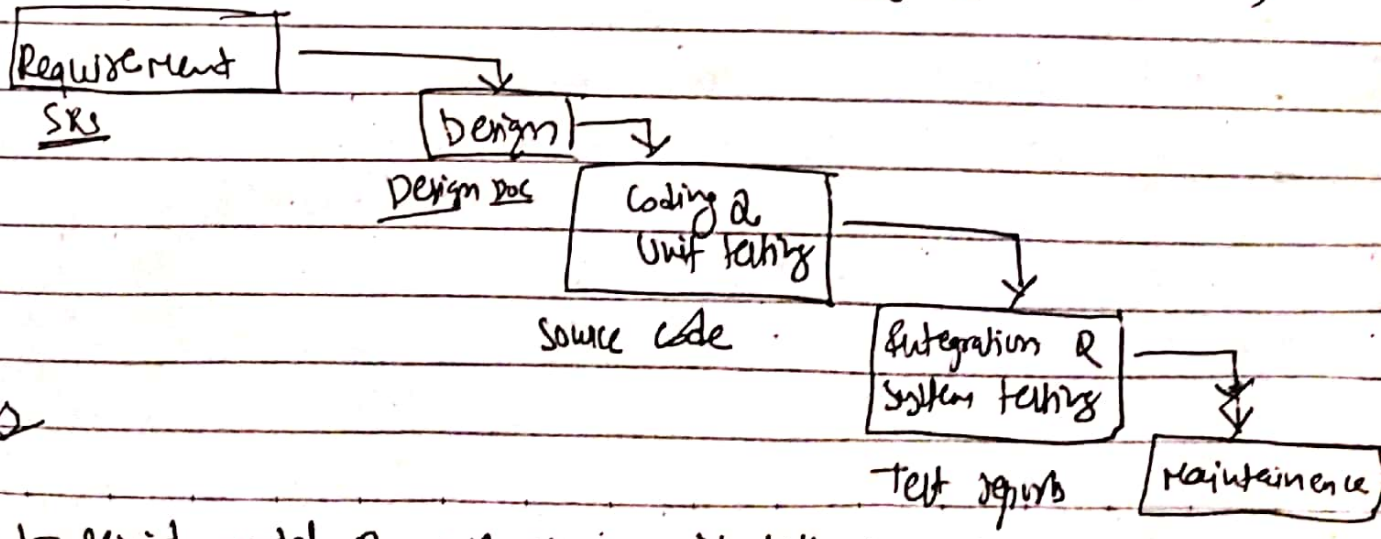
Diagramatic representation along with its Description. Documentation is done at every stage. as product is divided into sub modules.

Exit criteria - gt shows that a particular stage has been finished.
Once we finish (SRS (software requirement stage)) then we can say req. phase has been done.

99% complete syndrome - arises when phase entry & exit criteria are not defined clearly.

Classical waterfall model

feasibility study → take up a project (if feasibility study)



drawbacks

but Rigid model & we assume it follows a seq. path.

α- testing
β- testing
Acceptance testing

→ Deploy — maintainence
→ Collective - Everyone can report errors.
→ perfective maintainence - increasing scale of business
→ Adaptive Maintainence - Adapting new tech (updating software

SRS is used for documentation.

Iterative waterfall model

It is a improvent over waterfall model. phase
there is a path from any ~~model~~ to it's previous phase in case any error occurs.

Phase containment Errors —
we need to identify the error as soon as it is introduced.
milestones → if reached, reviews the work done.

drawbacks

1) Time consuming
2) Sequential model.
3) Underutilisation of resources.
4) risks are not handled properly.
Software failures

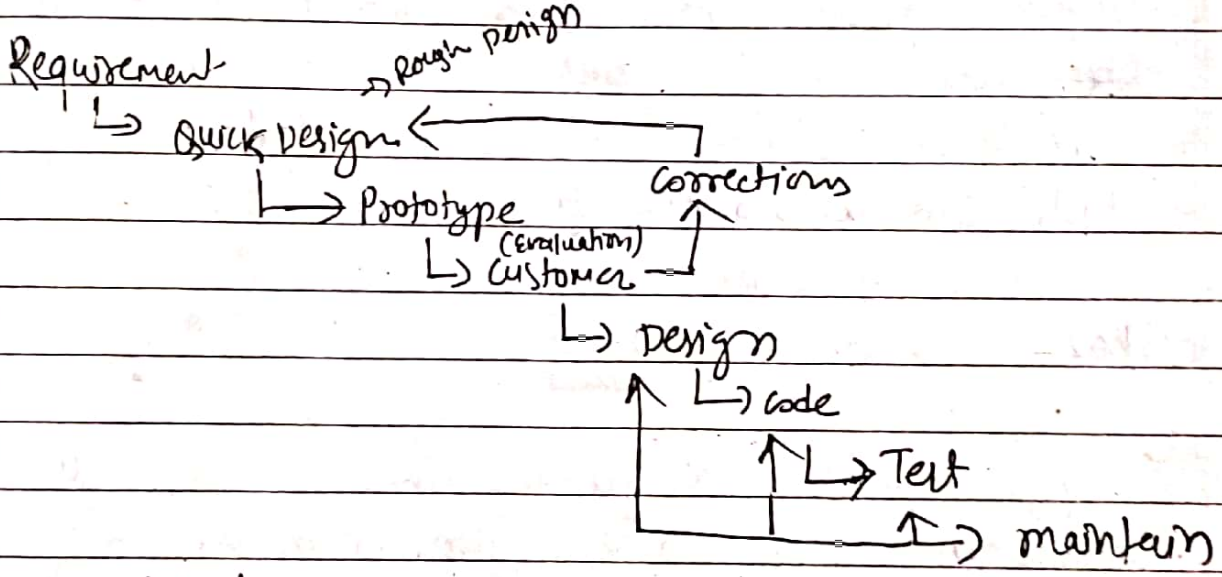
Adv.

→ basic & used for small projects
→ doesn't requires any experience

Prototype model

~~Require~~ ~~Requirements~~

Gives a first hand to the customer how exactly the product would be looking like.
for developer its cost. (frontend)



1) risk handling is done (using prototype)
2) Requirements need not be dead (even it is not clear)
3) If my staff does not has enough experience in developing this type of project.

Drawback

1) costly & time consuming

Evolutionary model / Incremental model

In incremental ~~model~~ model, we divide it in subparts & we add core modules and we will add functionality or features
final product is done by adding these all features.

Adv. -

1) ~~a~~ users doesn't have to wait for whole project to complete. We can provide some sub modules to the user ~~once~~ once it is finished
2) core modules would be throughly tested.
3) Risk can be handled easily

Disadv.

1) Applicable only to projects that can be further divided into subprog. (identifying core or independent modules may not possible every time)
2) ~~for every~~ every project ~~is~~ not possible

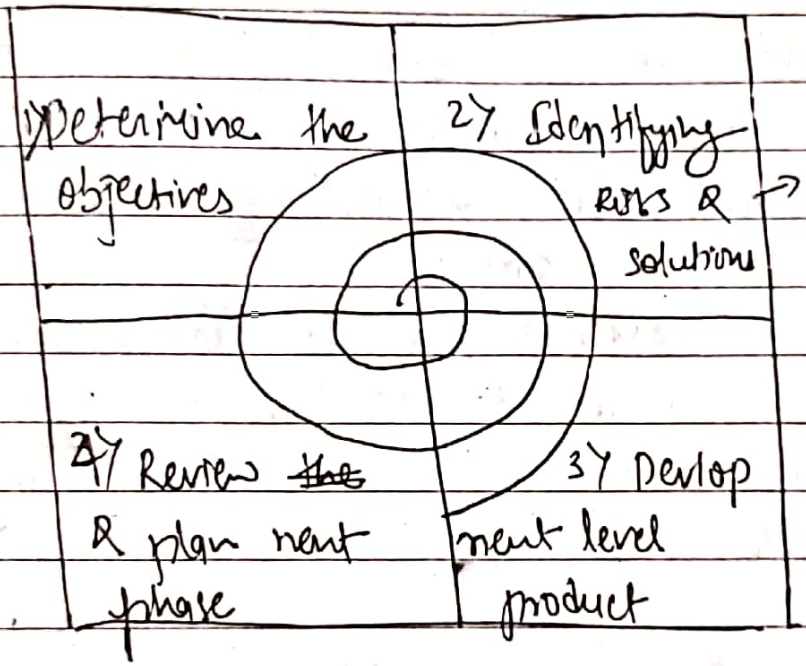
Rough requirements.
↓
Identify core modules & develop them incrementally
↓
develop core parts using iterative waterfall model
↓
Collect customer feedback & modify requirements
↓
Develop next identified features using iter waterfall method.

↓ all feature complete

maintenance

Spiral model

→ used for large & ~~medium~~ complex projects.
→ Risk management is ~~go~~ better than prev. models.
→ good customer interaction



for any project, we identify features / functions.
~~1) User registration~~ now for each feature, we will use a spiral.

disadv.

1) time consuming
2) ~~moving~~ back from phase to ~~phase~~ finished phase
2) It needs knowledgeable staff.
3) not suitable for outsource projects. (since risks are also transferred).

Spiral model as a meta model

Iterative waterfall model
prototype model
Incremental model
} all coated by spiral model.

When to use which Model

→ ~~IFM~~ - CWM - no use only for documentation
→ IWFM - for small projects & known projects
→ prototype Model - Medium & large proj. or inexperience members & medium risks
→ Incremental - large projects that can be divided into independent modules.

Agile Model or development

→ ~~aim to deli~~ Mixture of IWM & Incremental model. aim to deliver the right product, with incremental and frequent delivery of small chunks of functionality through small cross-functional self self-organizing teams, enabling frequent ~~and~~ customer feedback & course correction as needed.

→ It attempt to maximize the delivery of value to the customer and minimize the risks of building projects that do not or no longer meet market or customers needs.

→ Agile has become an umbrella term for a variety of planning, management and technical methods and processes for managing projects, developing software and other products and services in an iterative manner.

These methods include Scrum, by far the most prevalent and popular method for software, XP (extreme Programming) and lately Kanban

Scrum

→ projects are divided into cycles (typically 2 or 3 week cycles) called sprints. The sprint represents a timebox ~~for~~ within which a set of features must be developed.
Multiple sprints might be combined to form a release - where formal software / product delivery is made to the customer market.

→ The overall product functionality is broken down by the product owner into smaller features. These stories are prioritized and taken up in each sprint or iteration.

The intent of the method is for the team to be able to demo at the end of each sprint working pieces of the product to the product owner, to make sure that the project is working as intended.

Page No.
Date

Extreme programming (Paired programming)

It is characterized by developers working in pairs where one developer programs while the other developer observes. They switch the roles on regular basis throughout the sprint. This helps in better code generations

Kanban

→ It is a visual system for managing work as it moves through a process.
It visualizes both the process and the actual work passing through that process.
goal -
identify potential bottleneck in process & fix them.
for cost-efficiency at an optimal speed.

Why bother about req.
improper req →
→ leads to huge costs
→ increase the no. of iterative changes req. during life cycle phases
→ if not bothered, leads to customer dissatisfactions & may also lead to legal battles.

Why bother about requirements

→ improper requirement ~~changes the~~ increases the no. of iter. changes required during life cycle phases.

→ incorrect requirement leads to huge costs.

→ if not bothered leads to customer dissatisfaction & legal battles.

Goal of requirement specification :-

→ to clearly understand the customer requirements.

Systematically organize the requirements into specific document. The output is called software requirement specification (SRS).

Who does this :- ~~Pro~~ System analyst.

→ During analysis of req, one removes inconsistencies anomalies and incompleteness.

Some desirable attributes of a good system analyst :-

→ Experience
→ communication
→ Imagination & creativity.

Once SRS is done, it is given to the customer for review

once customer agrees, it forms the basis for all

future development activities and serves as a Contract-doctor document between the customer & the development organizations.

There are two activities involved in Requirements they are:-

→ Requirement gathering & analysis

Req. gathering:-

→ not a simple task

→ involves interacting with different types of customers who have diffrent level of understandability of the software & work.

Requirement gathering would be little easier if there exists a working model

→ it would be tedious if it has to be collected for a totally new project. Here, it would be a test for the analysts creativity & experience.

→ Requirement collected needs to be integrated & put it into proper meaning format.

Requirement gathering can be done in the following ways

→ Studying the existing documentation
→ Interview
→ Task analysis
→ Scenario analysis
→ Form analysis

Analysis of gather requirements:-

→ for clearly understanding user requirements
→ Detect inconsistencies, ambiguities and incompleteness
↓↓
resolved through further discussion with end-users & customers.

Inconsistent Requirement

→ Some part of requirement contradicts with some other part..

Incomplete Requirement
→ some req. have been omitted.
due to oversight.

Analysis of gather req:-
→ obtaining a clear, in-depth understanding of product to be developed.

→ Remove all ambiguities and inconsistencies from the initial customer perception of the problem.

Software requirement specifications (SRS)

→ highest, because it cater to the needs of a wide variety of audience (all stakeholders)

It is used in :-
→ statement for user needs
→ contract document
→ Reference document
→ Definition for implementation

Req. at this stage is written using end-user terms

Properties of good SRS

→ It should be concise & non-ambiguous.
→ It should specify what the system must do but not how to do it.
→ Easy to change & well-structured.
→ consistent & complete
→ It should be traceable & verifiable.
→ not is an & narrative essay format
→ does not contain noise & silence.
→ is not over specified ("how to")
→ There should be no contradictions & ambi

→ no wishful thinking & forward references.

→ SRS normally contains three things :-

→ functional requirements
→ non-functional requirement
→ Goals of implementation

functional requirements describe :-
→ A set of high-level requirements
→ Each high-level requirement
→ takes in some data from the user
→ outputs some data to the user.
→ might consist of a set of identifiable fn.

req-1 ← sub req.1
R1 R.1.1:
input:
output:
sub req.2
R1.2:

non-functional requirements

→ characteristics of the system which can not be expressed as functions:
→ maintainability
→ Portability
→ Usability

It includes :-

→ Reliability issues
→ Performance issues
→ human - computer interface issues
→ interface with other external systems
→ Security, maintainability, etc.
→ Capabilities of I/O devices
→ Hardware to be used
→ OS ~~to be used~~ / DBMS to be used
→ Standards compliance
→ Data representation

Goals of implementation

→ ~~Retat~~ Reusability issues
→ functionalities to be developed in future

documentation of Functional requirement
→ It is done by identifying the state at which the data is to be input to the system
→ its input data domain
→ its output data domain
→ the type of processing to be done..

Ajanta
Page No.
Date

Or → Introduction
→ functional requirements
→ Non-functional requirements
→ External interface requirements
→ Performance requirements
→ Goals of implementation

} Organization of SRS Document

Handling complex logics :-

→ The complex processing is analysed in two ways
→ Decision trees
→ Decision tables

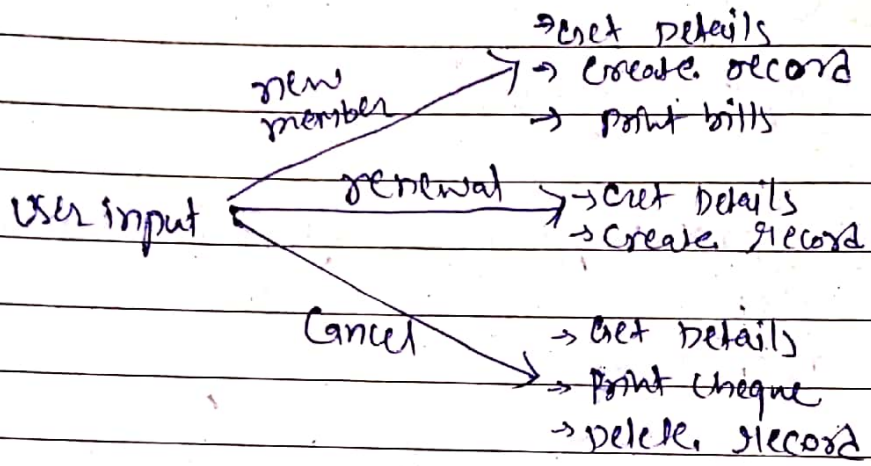
→ A decision tree gives a graphic view of the processing logic involved in decision making
→ A decision table shows the decision-making logic and corresponding actions taken in a ~~matrix~~ matrix form.

Decision trees

→ Edges of a decision tree represents conditions
→ leaf nodes represents actions to be performed.

It gives a graphic view of :-
→ logic involved in decision making.
→ ~~Cor~~ Corresponding actions taken.

ex - a lib. membership automation software. Should support the following three options :-
→ new member
→ Renewal
→ Cancel membership.



Decision table →

It specify :-
→ which variables are to be tested.
→ what actions are to be taken if the conditions are true
→ The order in which decision making is performed
→ A decision table shows in a tabular form.
→ upper rows of table specify:
- The variables or conditions to be evaluated
→ lower rows specify :-
- The actions to be taken when the corresponding conditions are satisfied.

→ a column of the table is called a rule.

→ A rule implies:

→ if a condition is true, then execute the corresponding action.

Conditions				
A	no	yes	yes	no
:				
:				
:				
Actions				
x				
y				

both decision trees & decision tables represent complex program logic but decision trees are easier to read & understand, when numer of conditions are small. but decision tables help to look at every possible combination of conditions.

→ formal specification → for finding acute error

→ it is a mathematical method to :

→ Accurately specify a system

→ verify that implementation satisfies specification.

→ prove properties of the specification.

Advantages :

→ well-defined semantics, no scope for ambiguity

→ Executable specifications

→ Automated tools can check properties of specifications.

Disadvantages of formal specifications
→ difficult to learn & use
→ not able to handle complex systems
→ Axiomatic technique - uses first order predicate logic (uses pre & post conditions)
Algebric specification -

→ data types are viewed as heterogenous algebra
→ axioms are used to state properties of data type operations.

Using algebraic specifications - the meaning of a set of interface procedures is defined by using equations

It contains four sections
→ Types section
→ Exceptions section
→ Signature section
→ Rewrite rules section

Types section defines :-
→ data type being used
→ types being imported
→ Importing a sort :
makes it available for specifications.

Exception Section -
→ The conditions where exception may occur

Signature Section

Defines the signatures of interface procedures :
Ex - push takes a stack & an element and returns a new stack
→ push : Stack x element → stack

Rewrite rules Section :
→ in the form of a set of axioms or rewrite rules
→ allowed to have conditional expressions

First step of defining an algebraic specification :
→ identify the set of required operations.

Ex - for string identify operations - Create, compare, etc.

identify operation type

(i) Constructor operation - creates or modifies entities
(ii) Inspection operation - evaluates attr. of entities

first establish const. then insp. operation.

If a constructor operation can be defined using other constructors:

→ We need to define inspection operations using only primitive constructors.

ex - stacks

Types :
defines stack
uses boolean, element

Exceptions :
underflow, novalue

Syntax : → (signature)

Push:
- Stack x element → stack

pop: → exceptions
- stack → stack + {underflow}

newstack:
- ϕ → stack

top:
- stack → element + {novalue}

Equations :-

pop(newstack) = underflow
pop(push(s,e)) = s

Specification Instantiation

→ Specified with some generic parameter
→ instantiate with some sort (type)

Incremental Development

Use specifications for simple sort & use these to Build more complex entities

Algebra Specifications pros & cons

→ have strong mathematical basis
→ can be viewed as heterogenous algebra

disadv.

→ Cannot deal with side effects.
→ difficult to use with common programming lang.
→ hard to understand
→ also changing a single property of the system may require changing several equations

Specifications of timing constraints

→ expressed in terms of occurrence of certain events.

Event

→ A stimulus to the system from its environment
→ can also be an externally observable response that the system makes to its environment
→ It can be instantaneous or assumed to have a duration.

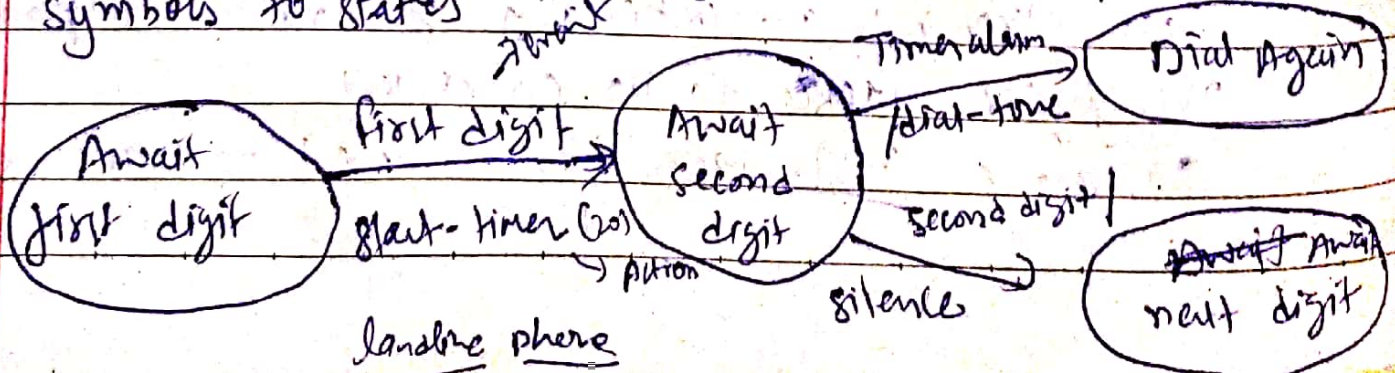
Types Timing constraints are of two types :
Performance constraints :
→ constraints imposed on the response of the system

Behavioral constraints
→ constraints imposed on the action and reaction time of the environment

We will specify timing ~~Constrai~~ constraints :
→ in terms of stimuli & response ($S^{I/P} \rightarrow R^{O/P}$)
→ modelled as FSMs (finite state machines)

State machine modelling
→ it assumes that ~~an~~ at any time the system is in one of a number of possible states
→ when a stimulus is received → it may cause a transition to a different state.

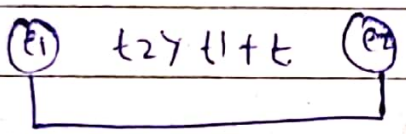
finite automata with O/P
→ set of states having some states as final states
→ an alphabet of input symbols
→ an alphabet of output symbol
→ a transition function
→ maps a combination of state and input symbols to states



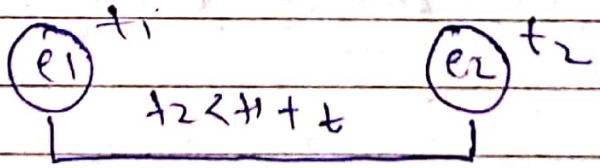
a finite automation with o/p may be organized in two ways:
→ a moore machine is an automaton in which each state is associated with an output symbol
→ a mealy machine associates each transition with an output symbol

-three types of timing constraints:-

i) Minimum
between two events no less than t units may elapse



ii) maximum
between two events no more than t time units may elapse



iii) Durational
An event must occur for t units of time

maximum S-5
the maximum time between occurrence of two stimuli
eg - for dialing a first digit, the second digit should be dialed no more than 20 secs later.

Similary Maximum SR is maximum time between Stimulus and response.

Maximum SS event and action maximum RS is the maximum time between response & next stimulus from the environment.

Similary, minimum constraints
S-S - Minimum time b/w two stimulous.
R-S - minimum " " Response & stimulous
R-R - " " " Response & Response.

Durational timing Constraints -
→ pressing button for 30 seconds.

Software project Management

many software project fails due to faulty project management practices. So, it is important to learn different aspects of Software project Management

It helps in enabling a group of engineers to work efficiently towards successful completion of a software project.

Page No.
Date

Responsibility of project Managers :

→ Project proposal writing
→ Project cost estimation
→ Scheduling
→ Project staffing
→ Monitoring & control
→ Risk Management
→ Software configuration management
→ Managerial report writing & presentations etc.

Responsibility of Project Manager

Project planning

Project monitoring & control activities.

If project is found feasible then project Managers * undertake project planning.

Project planning Activities :

→ Estimation - effort, cost, resource and project duration
→ Project scheduling
→ Staffing plans
→ Risk handling - identification, analysis & abatement procedures
→ Miscellaneous plans -
quality assurance plan, configuration management plan, etc.

Errors or mistakes in project planning leads to :-
→ irrational delays
→ customer dissatisfactions
→ adverse effect in team morale i.e poor quality work
→ project failure

→ Sliding window planning :-

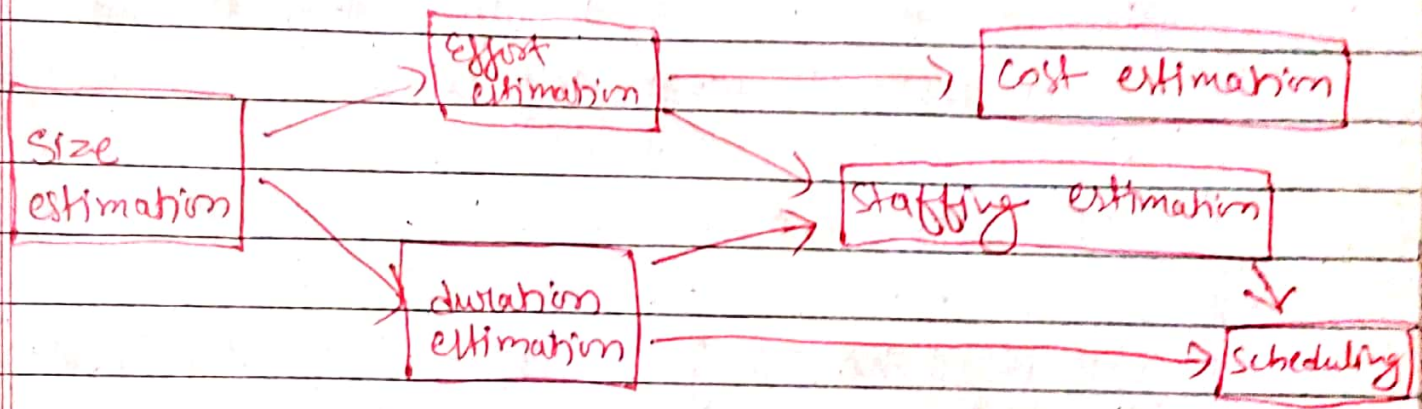
→ involves project planning over several stages.
→ protects managers from making big commitments too early.
→ more information becomes available as project progresses i.e. facilitates accurate planning.

Software project management plan document :-
is used to plan the document.

Organization of SPMP document :-
→ introduction
→ objectives, major functions, performance issues, management and technical constraints
→ Project Estimates
→ Project resources plan
→ schedules
→ Risk management plan
→ Project Tracking and control plan.
→ Miscellaneous plans

Software cost estimation :-

→ Determine size of the product
  → from this, determine the effort needed.
→ determine project duration & cost.



Software size metrics :-

→ LOC (Lines of code) :
  → Simplest & most widely used metric
  → Comments and blank lines should not be counted

Disadvantages of using LOC -
→ size can vary with coding style.
→ focuses on coding activity alone.
→ correlates poorly with quality and efficiency of code.
→ Penalizes higher level programming languages, code reuse, etc
→ measures lexical / textual complexity only
  → does not address the issues of structural or logical complexity.
→ difficult to estimate LOC from problem description :
so not useful for ~~pr~~ project planning.

function point metric

→ overcomes some of the shortcomings of the LOC metric.

Proposed by Albrecht in early 80's:

UFP = 4 x #inputs + 5x #outputs + 4x #inquiries + 10x #files + 10x # interfaces

FP = UFP x TCF

⇒ ufp is further refined using the refinement of function point table.

Refinement of function point entities table

Type	Simple	Average	Complex
input	3	4	6
Output	4	5	7
Inquiry	3	4	6
Number of files	7	10	15
No. of interfaces	5	7	10

function point metric

→ Albrecht identified 14 parameters that influence the development effort.

→ These 14 parameters are assigned a value from 0 to 6.

the resulting sum is called degree of influence.

TCF = (0.65 + 0.01 * DI)
Where.

finally FP is computed as UFP * TCF

Input - A set of related inputs is counted as one input.
Output - A set of related outputs is counted as one output.
Inquiries - Each user query type is counted.
Files - files are logically related data thus can be data structures or physical files.
Interface - Data transfer to external systems.

Pros → It is language independent
→ Size can be easily derived from problem description.

Cons →
→ Size of a function is considered to be independent of its complexity
→ It is subjective. Different people can come up with different estimates for the same problem.

Soln → Extended function point metric :
Feature point metric : Considers an extra parameter i.e Algorithm complexity.

Page No.
Date

Software Cost estimation :-

Three main approaches :-

→ Empirical –
an educated guess based on past experience.

→ Expert judgement
→ uses a expret guess for cost estimation
→ suffer from human error & individual bias

→ Delphi estimation → (experts re-estimate same product)
→ overcomes some problems of expert judgement
→ uses multiple judges & takes their judgement seperatly (~~gives~~ a coordinater does this).

→ Heuristic Estimation techniques :-
assumes that the characteristics to be estimated can be expressed in terms of some mathematical expression.

→ Single variable model :-
Parameter to be Estimated = C_1 (Estimated characteristic)^{d_1}
Estimated parameter = $\Theta\ c_1 * e^{d_1}$

→ multivariable Model :-
→ Assumes that the parameter to be estimated depends on more than ~~one~~ one characteristic.

parameter to be estimated = C1 (estimated characteristic) d_1 + C2 (estimated characteristic) d_2 + ...

Estimated resource = $c_1 * ep^{d_1}_1 + c_2 * ep^{d_2}_2 + \ldots$

→ more accurate than single variable models.

COCOMO Model

Constructive Cost Model was proposed by Boehm.

divides software product developments into 3 ~~Categories~~ Categories :-

i) organic :- Relatively small groups of working to develop well-understood applications.

ii) semidetached - project team consist of a mixture of experienced and inexperienced staff.

iii) embeded - the software is strongly coupled to complex or real time systems.

Cocomo product classes Roughly correspond to :-

→ application, utility & system programs respectively

→ Data processing & scientific programs are considered to be application programs

→ compilers, linkers, editors etc. are utility programs.

→ OS & real-time system programs are system programs.

Page No.
Date

fore each of the three categories → from size estimation (in KLOC), Bohem provides equations to predict :-

→ project duration in months
→ effort of programmers in math-months

Cocomo model

Software cost estimation can be done in 3 stages :-

→ Basic Cocomo :-
→ Gives only approximate estimation.

$$\text{Effort} = a1 \times (\text{KLOC})^{a2} \text{ PM}$$
$$\text{Tdev} = b1 \times (\text{Effort})^{b2} \text{ months}$$

KLOC - kilo loc

a1, a2, b1, b2 are constants for different categories of software products.

Tdev → estimated time to develop the software in months.

Effort estimation is obtained in terms of person months (PM)

cost estimation

Organic :- $\text{Effort} = 2.4 (\text{KLOC})^{1.05}$ PM

Semi-detached :- $\text{Effort} = 3.0 (\text{KLOC})^{1.12}$ PM

Embedded :- $\text{Effort} = 3.6 (\text{KLOC})^{1.20}$ PM.

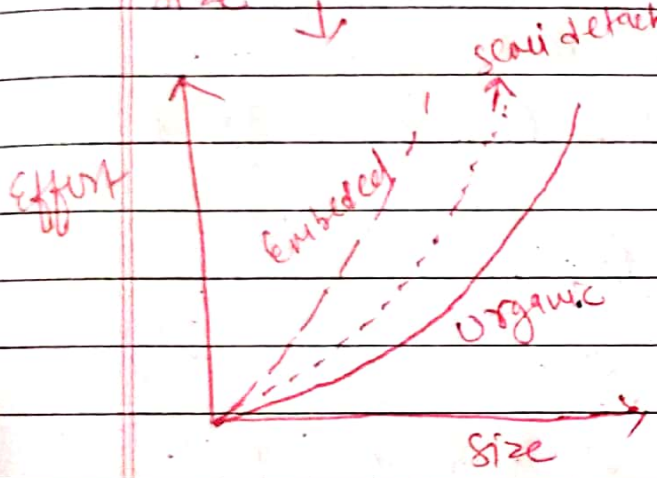
Time estimation

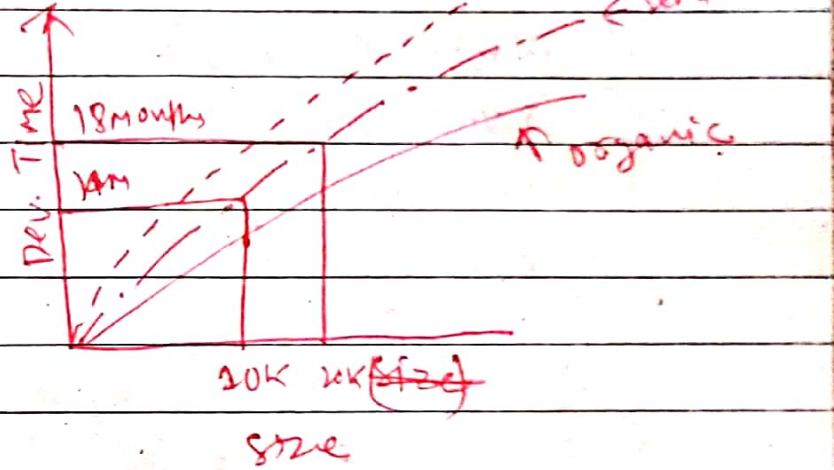
organic :- $T_{dev} = 2.5 (Effort)^{0.38}$ Months

Semi-detached :- $T_{dev} = 2.5 (Effort)^{0.35}$ Months

Embedded :- $T_{dev} = 2.5 (Effort)^{0.32}$ Months

Basic Cocomo effort is somewhat super linear to size





When product size increases 2 times development time does not double.

Effort time taken by all the three categories are almost same

Development time does increase linearly with product size because of parallel activities & more reusable code.

Page No.
Date

Intermediate cocomo (multivariate model)

→ Basic cocomo model assumes
→ effort and development time depend on product size alone but it is not so.

→ factors affecting effort & development time:-

→ Retat Realiability requirements
→ Availability of CASE Tools & modern facilities to the developers
→ Size of data to be handled.

for accurate estimation
→ the effect of all relevant parameter must be considered.

intermediate cocomo model recognizes this fact refines the initial estimate obtained by the basic cocomo by using a set of 15 cost drivers (multi

→ if modern practices are used, initial estimates are scaled downwards
→ if there are stringent reliability requirement on the product then initial estimate is scaled upwards.

rate different parameter on the scale of one to three Depending on the these ratings multiply cost drive

values with the estimate obtained using the basic Cocomo.

Cost driver classes :
→ Computer - Execution time, storage requirement, etc.
→ Personnel - Experience of personnel, etc.
→ Product - Inherent complexity of the product, reliability requirements of the product, etc.
→ Development Environment : Sophistication of the tools used for software development.

Shortcomings of basic & intermediate Cocomo Models :

→ They consider software product as single homogeneous entity however, most large systems are made up of several smaller sub-systems. Some sub-systems may be considered as organic type, some ~~called~~ as embedded, etc.

→ for some the reliability reliability may be high & so on.

Complete Cocomo

→ Cost of each subsystem is estimated separately.
→ Cost of sub systems are added to obtain total cost.
→ reduces the margin of error in the final estimate.

Eg: like - in MIS database part, GUI & communication part.
semi-detached organic communication part

Page No.
Date

Cost of components are seperately estimated & summed up to give overall cost of the system.

Provides 3 increasingly detailed cost estimation models:

→ Application composition - prototyping
→ early design - higher design stage
→ Post-architecture stage - detailed design & coding stage

Application composition model

→ Effort is estimated as follows :
→ No. of screens, reports, 3GL components of system
→ Determine complexity level of each screen based on following table :

No. of views	Tables < 4	Tables < 8	Tables >= 8
<= 3	simple	simple	medium
3-7	simple	medium	difficult
> 8	medium	difficult	difficult

Similarly complexity level of each report complexity :

No. of sections	Tables < 4	Tables < 8	Tables >= 8
0 or 1	simple	simple	medium
2 or 3	simple	medium	difficult
4 or more	medium	difficult	difficult

→ Determine no. of object points
→ summation of complexity values for a object.

→ Estimate the percentage of reuse expected in a system and compute new object points. (NOP)

Determine the productivity rate, Prod = NOP/PM
PM is computed as ~~Prod~~ NOP/Prod.

Early design Model
Seven cost drivers that characterize the post-architecture model are used.
effort = KSLOC X PL; costdriver;

Post architecture model

Differs from original Cocomo model by the choice of the cost drivers
effort = a X KSLOC^b X PL; costdriver;
where b ranges from 1.01 to 1.26

Date

Halstead's Software Science

an analytical technique to ~~understand~~ estimate :-
→ Size
→ development effort
→ development time

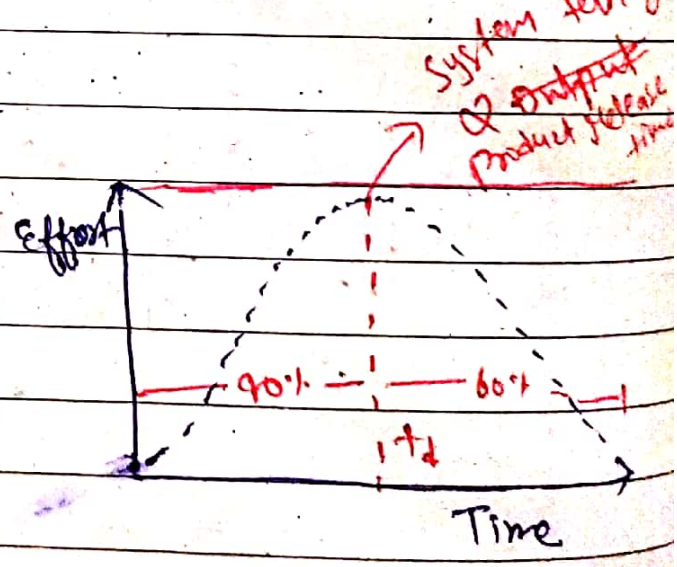
Halstead used few primitive program parameters :-
→ no. of operators and operands

Staffing level estimation

no. of personnel required during any development project is not constant.

Rayleigh Curve

It represents the no. of full time personnel required at any time.



It is specified by two parameters
t_d → the time at which the curve reaches its maximum
K → total area under curve.

$L = f(K, t_d)$

Very small Engineers are needed at the begining of a project to carry out planning & specification.

As project increases or progresses more detailed work is required the number of engineers slowly increases and reaches a peak. Then, it decreases.

Putnam work :-

$$L = C_k K^{1/3} t_d^{4/3}$$

Where, K is the effort expended & L is the size in KLOC.
t_d is the time to develop the software.
C_k is the state of technology constant.
Reflects factors that affect programmers productivity.

$C_k = 2$ → poor development environment
→ no methodology, poor documentation, & reviews, etc.

$C_k = 8$ good software development Environment
→ software engineering principles used.

$C_k = 11$ for excellent environment

Effect of schedule change on cost

Using putnam's expression for t,

$$K = L^3 / (C^3 t^4)$$

or $K = C / t_d^4$ where $C = L^3 / C_K^3$

So,

$$\frac{K_1}{K_2} = \frac{t_{d2}^4}{t_{d1}^4}$$

→ a relatively small compression in delivery schedule can result in substantial penalty on human effort.

⇒ Benifits can be gained by using fewer people over a somewhat longer time span.

Ex - If estimated dev. time of a product is 1 year & in order to develop the product in 6 months the effort & cost increases 16 times.

Extreme penalty for schedule compression & Extreme reward for Expanding the schedule

Work well for very large systems & but seriously overestimates the effort for medium & small systems.

The maximum limit of time reduction is 75%.

Jensen Model

→ very similar to putnam model :-
→ attempts to soften the effect of schedule compression on effort.
→ makes it applicable to smaller and medium sized projects.

Jensen proposed :-

$$L = C_{te} t_d K^{1/2}$$

where, C_{te} is effective technology constant
t_d is time to develop the software &
K is the effort needed to develop the software.

Project Scheduling

→ involves deciding which tasks would be taken up when.
→ To schedule project following things are needed to be done :-

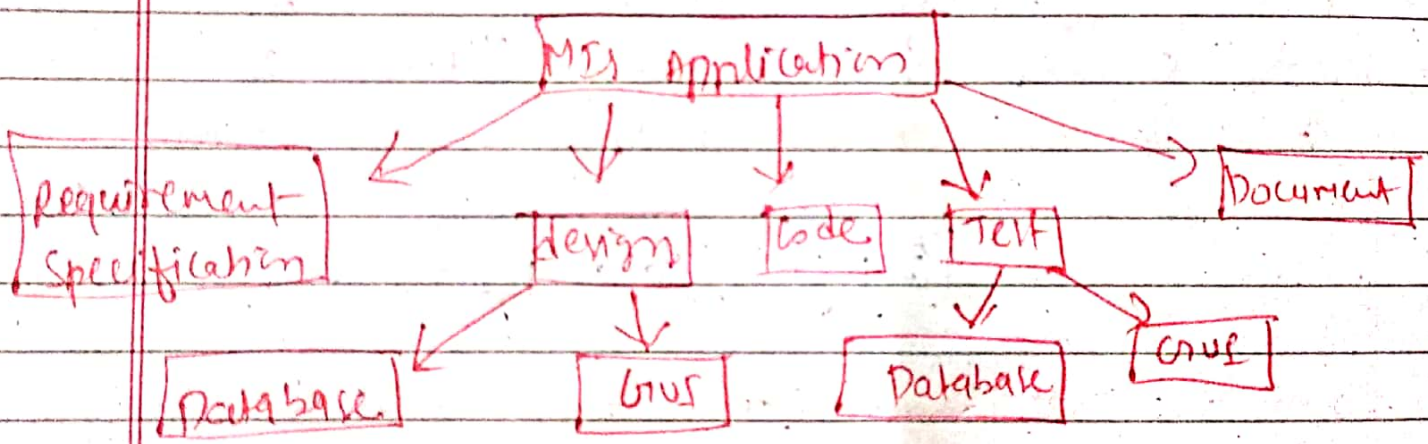
→ identify all the things needed to complete the project
→ Breakdown large task into small activities.
→ Determine the dependency among diffrent activities.
→ establish the most likely estimates for the time durations necessary to complete the activities
→ Allocate resources to activities.
→ plan the starting & ending dates for various activities.

→ determine the critical path.
A critical path is the chain of activities that determines the duration of the project.

Work Breakdown Structure

→ It is used to decompose a given task set recursively into small activities

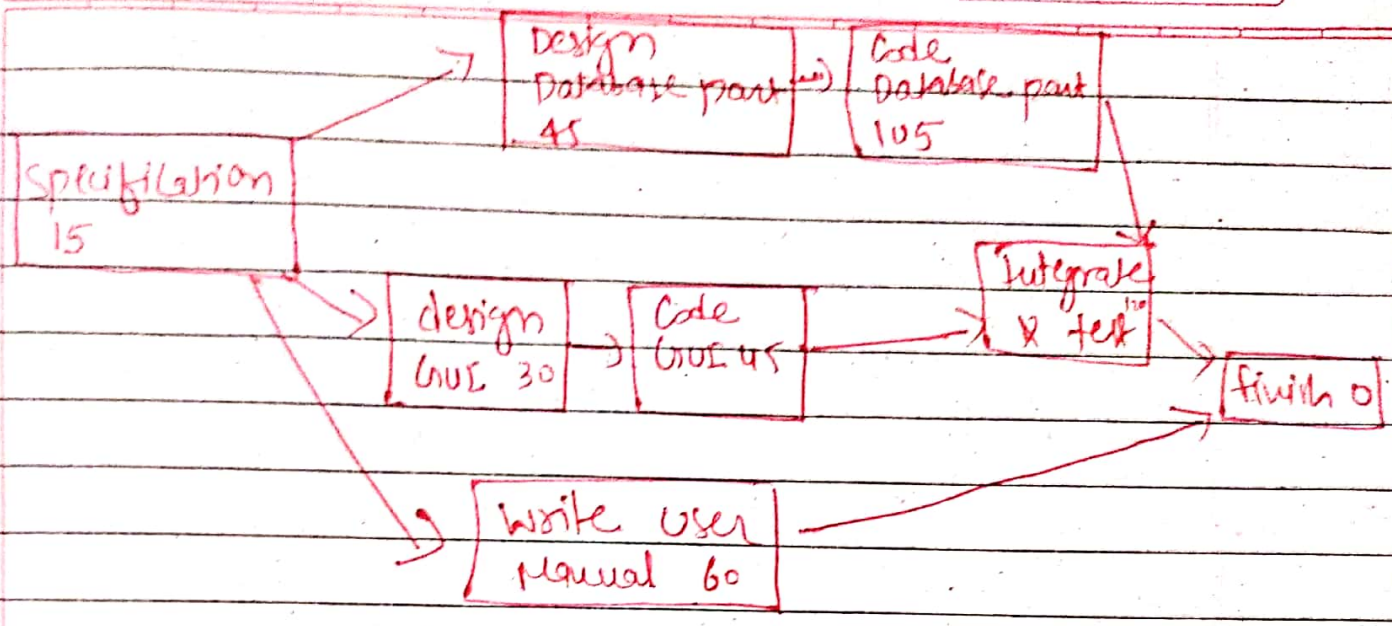
→ Root of the tree is labelled with the problem name.



Activity networks

It shows the different activities making up the project their estimated durations & interdependencies.

Each activity is represented by a rectangular node & the duration of activity



Critical path Method (CPM)

It is used to find the numerical way we can schedule the activities of a project.

Terms in finding the CPM :-

→ MT (Minimum time) - max of all paths from start to finish to complete a project.
→ ES (earliest start) - max of all paths from start to the task
→ LS (latest start) - difference between MT & max of all paths from this task to finish